ARMY RESEARCH LABORATORY

Mobile Ad-Hoc Networking on Android Devices

by Felipe Jovel, David Bruno, David Doria, Jonathan Fletcher, Tamim Sookoor

ARL-TR-6845 **March 2014**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory

Aberdeen Proving Ground, MD 21005-5066

ARL-TR-6845 **March 2014**

Mobile Ad-Hoc Networking on Android Devices

David Bruno, David Doria, Tamim Sookoor
Computational and Information Sciences Directorate, ARL

Filipe Jovel
University of Texas at El Paso
El Paso, TX 79902

Jonathan Fletcher
Secure Mission Solutions
North Charleston, SC 29406

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
March 2014	Final	March 2014

4. TITLE AND SUBTITLE
Mobile Ad-Hoc Networking on Android Devices

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Felipe Jovel,* David Bruno,David Doria,Jonathan Fletcher,† Tamim Sookoor

5d. PROJECT NUMBER
R.0006163.13

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-CIS-H
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-6845

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
*University of Texas of El Paso, El Paso, TX 79902
†Secure Mission Solutions, North Charleston, SC 29406

14. ABSTRACT
Mobile devices provide civilians with a convenient way to communicate on a day-to-day basis. However, their centralized infrastructure can be interrupted at critical times such as during natural disasters. While this cellular infrastructure may not be available in environments, such as the battlefield, these ubiquitous devices could exploit ad-hoc mode communication to be of use. The Army has identified several military applications for ad-hoc networks of mobile devices, including communication between dismounted Soldiers, facial recognition, and situation awareness. Many of these applications require more capabilities than most mobile devices possess, or would prove to be a burden on the limited battery lives of these devices. Thus, offloading computation to a more capable machine has been identified as a means to make these devices useful while mitigating their weaknesses. This has lead Army researchers to develop novel algorithms to improve the offloading process. Although these algorithms have been tested via simulation, a working implementation is necessary to fully evaluate them. This report discusses the state of ad-hoc mode communication on Android devices, and describes the procedure for enabling ad-hoc mode, which is often not available by default. It goes on to describe techniques for studying multiple aspects of the network and its behavior.

15. SUBJECT TERMS
mobile ad-hoc network, android

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Filipe Jovel
Unclassified	Unclassified	Unclassified	UU	28	19b. TELEPHONE NUMBER (*Include area code*) 410-278-8929

Contents

List of Figures

INTENTIONALLY LEFT BLANK.

1. Introduction/Background

1.1 Ad-Hoc Networks

According to the online Merriam Webster dictionary, ad-hoc comes from Latin, meaning "for this" and is usually used in the context of doing something for a particular end or purpose. For example, when a file is transferred between two devices, a network for the purpose of transferring the file without any other hardware but the two devices involved in the transfer is created. This is in contrast to the devices operating in "infrastructure mode," meaning that their activity is coordinated by a basestation (a wireless router, etc.). In infrastructure mode, when a device wants to communicate with any other device except the master, it is forced to do so through the master device, which forwards the message to the destination. Likewise, when the recipient of the message replies, its message will have to travel through the master to eventually get back to the sender. This technique carries two inherent problems. First, since all traffic has to travel through the master device, the master device may get overloaded when several clients want to continuously send information to each other. The second problem is that having a master device introduces a single point of failure. If the master device stops working, then the entire network below it disappears. Both of these problems could be avoided if the devices could talk directly with other devices in range.

From their inception mobile ad-hoc networks, ad-hoc networks of mobile devices, were meant for aiding Soldiers in combat operations. The Defense Advanced Research Projects Agency (DARPA) developed the very first mobile ad-hoc network in 1973 called Packet Radio Network (PRNET). PRNET was a first attempt to solve the problem of being unable to communicate when no infrastructure is present, as is often the situation in the battlefield. PRNET made use of a common radio channel to transfer data between computer nodes that may be actively moving. More recently, catastrophic events such as Hurricane Katrina (2005), Haiti's Earthquake (2010), and Fukushima's Earthquake/Tsunami (2011) have shown us that there are many uses for infrastructure-less communication even beyond the battlefield. For example, during Hurricane Katrina over 3000 cell sites in the Mississippi Coast, New Orleans, and Plaquemines Parish areas were affected, causing severe interruptions to the mobile communication network. It took a week to bring the mobile communication network back to almost full operation along the Gulf coast, and partial operation in New Orleans and Plaquemines Parish (*1*). A week might seem like a considerably short duration for the magnitude of such an event, however, it is a very long time

when a disaster has occurred and citizens need to communicate with their loved ones to let them know they have survived, or when they need to call for potentially life-saving help.

It is clear that a solution to infrastructureless communication is needed. PRNET was a good first step; however, the Packet Radios technology has grown old. At the end of 2011 there were six billion mobile cellular subscriptions (*2*). This was about 86% of the world population in 2011, and this number does not even include mobile devices with Wi-Fi capabilities. With mobile devices becoming so popular and present everywhere, they have been targeted to become the next platform for mobile ad-hoc networks. Thus, a robust ad-hoc mode communication capability is essential for mobile devices.

1.2 Army Application — Computation Offloading

The Army is working to empower the Soldier with mobile devices capable of creating and maintaining a mobile ad-hoc network. Furthermore, due to the limited battery and computation capabilities of mobile devices, research is being done to take advantage of High Performance Computer (HPC) resources seeded within the mobile ad-hoc network. Having access to HPC resources allows the offloading of compute-intensive tasks, thus reducing the power consumption of mobile devices as well as their time-to-solution. Computation offloading is essential to make ad-hoc networks of mobile devices viable tools on the battlefield, and thus, various approaches to optimize it are being explored.

Novel offloading algorithms being developed by Army researchers have been tested in simulation (*3*). However, experimental studies with actual hardware are essential to validate the results provided by these simulations (*4*). In order to develop tools to validate these algorithms, we first needed to identify a way of constructing a mobile ad-hoc network on mobile devices.

1.3 Rationale for Mobile Platform Selection

During the third quarter of 2012 Android and iOS devices dominated the market share with 75.0% and 14.9%, respectively (*5*). This strongly suggested Android devices as our platform of preference. Not wanting to discard iOS without any evaluation, we took a closer look at both platforms. Although iOS has the ability to join ad-hoc networks, it lacks the ability to create them. This lack of ability and the fact that the Android operating system is open source, while iOS is very restrictive and closed source, was our determining factor for choosing Android. Android's openness proved to be a key feature to successfully creating a mobile ad-hoc network.

2. Wireless Technologies

There are three primary technologies that enable ad-hoc communication between devices: Bluetooth, Wi-Fi Direct, and Ad-hoc Mode of the IEEE 802.11 standard. Currently, Android has native support for Bluetooth and Wi-Fi Direct ad-hoc modes, but not for the IEEE 802.11 standard. In this section we briefly discuss these technologies and their applicability to our problem.

2.1 Bluetooth

Bluetooth is a short-range (100 m) low-data rate (3 Mbps) wireless technology capable of forming ad-hoc networks (*6*). Bluetooth's ad-hoc network basic topology is a piconet implementing master/slave architecture. A piconet consists of one device assigned as master, and up to seven slave devices that together form an ad-hoc network. A diagram of a piconet is shown in figure 1. A device may act as a slave in multiple piconets, in which case the device is part of a scatternet. A scatternet (shown in figure 2) is an ad-hoc network formed by joining multiple piconets.

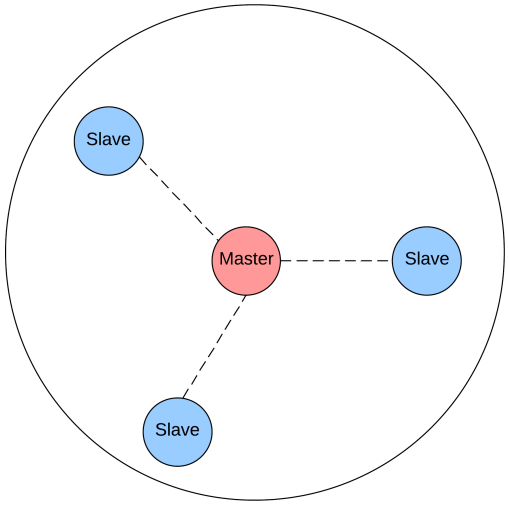

Figure 1. A piconet.

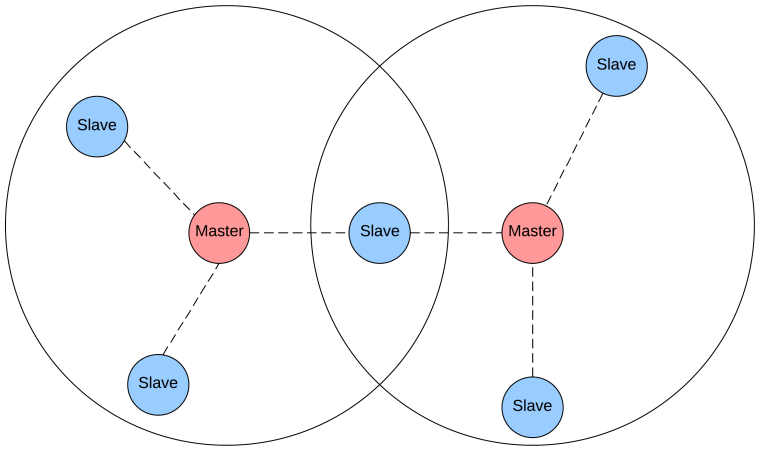

Figure 2. A scatternet.

Due to the Army's requirements of relatively long-range communication (kilometer scale) and high-data rate transfer capabilities, this technology is not appropriate for our needs.

2.2 Wi-Fi Direct

Wi-Fi Direct is the branded name for the specification "Wi-Fi Alliance Peer-to-Peer" developed by the Wi-Fi Alliance (*7*). Wi-Fi Direct works on top of the 802.11n standard, which has a range of 250 m and a data rate of 248 Mbps (*6*). Like Bluetooth, Wi-Fi Direct implements a master/slave architecture. A Peer-to-Peer (P2P) Group is a Wi-Fi Direct ad-hoc network. A P2P device is a device that follows the Wi-Fi Alliance P2P specification. A P2P Group is capable of supporting both P2P devices and traditional Wi-Fi devices. In a P2P Group, P2P devices can take the role of a group owner (master), or P2P client (slave). Legacy, or traditional, Wi-Fi devices are only capable of taking the role of P2P clients in a P2P group. P2P groups require only a P2P group owner to exist. When a P2P device creates a P2P group, it acts as a software defined access point, and P2P clients act as stations of this access point, similar to how infrastructure wireless networks operate. The P2P specification mentions that devices may form part of two different Media Access Control (MAC) entities. This suggests that similar to Bluetooth, one could extend a P2P group network by having one P2P device act as a client in two different groups. However, the specification for Wi-Fi Direct does not go into any more detail on the matter.

Essentially Wi-Fi Direct is only enabling a "faux-ad-hoc" network. That is, it is allowing devices to communicate without the need for a dedicated access point, but it still requires the data to flow through a particular node "acting" as the access point. For this reason, we have not selected Wi-Fi Direct as the technology to use for our problem.

2.3 Ad-Hoc Mode of the IEEE 802.11 Standard

After concluding that Bluetooth and Wi-Fi Direct are not suitable for our needs, we focused our efforts on trying to set up a network using the IEEE 802.11 ad-hoc mode, or Independent Basic Service Set (IBSS), on Android devices. Unfortunately, Android devices do not support IBSS natively. In this section, we discuss how to enable IBSS support on Nexus 7 devices.

With IBSS support enabled, Android devices can communicate directly with each other as long as they are within radio range of each other. There is no need for a device to act as an intermediary between devices communicating, as in the case of Bluetooth and Wi-Fi Direct.

2.3.1 Device Compatibility

Android 4.x device implementations are required to include support for one or more forms of 802.11 (b/g/a/n, etc.). However, not all Android devices are IBSS-capabile out-of-the-box. Some devices will fail to discover other IBSSs. In this section, we detail the procedures required to enable IBSS on several different Android devices, and comment on their suitability to be used in an ad-hoc networking environment.

Asus Nexus 7 Out-of-the-box, the Nexus 7 is not able to see any IBSSs. The culprit of this missing functionality is the wireless chipset driver. Although support for IBSS exists in the hardware, the driver disables it. In order to enable IBSS, the kernel driver for the wireless chipset must be modified. The kernel has to be recompiled with the changes, and the modified kernel uploaded to the device. Since all Android devices do not use the same wireless chipset, these modifications may vary from device to device. When attempting to connect to a wireless network, the first step is to request the kernel driver to search for neighboring Basic Service Sets (BSSs) or IBSSs. The original chipset driver filters IBSS networks, and this is why the Android is unable to see IBSS networks in the scan results. In Code Listing 1 we show an excerpt 'diff' output of the original and modified driver file. This is the main modification needed for Broadcom chips, which is well known to the Android developer community.

Code Listing 1. A diff excerpt demonstrating how to enable IBSS on Android devices using Broadcom drivers. The line starting with the minus sign is replaced with the line starting with a single plus.

```
+++ b/drivers/net/wireless/bcmdhd/wl_cfg80211.c
- BIT(NL80211_IFTYPE_STATION)
+ BIT(NL80211_IFTYPE_STATION) | BIT(NL80211_IFTYPE_ADHOC)
```

When experimenting with the Nexus 7 devices, we noticed sporadic outages of communication between the devices. After consulting with the authors of the patches to the driver required to enable ad-hoc mode, we were informed that there is an unresolved issue that they call "cell splitting." Typically, the desired behavior is for a new device to join an existing ad-hoc network if one is found. With the cell splitting problem, a device would erroneously connect to a different BSSID (Cell) than other devices in an already established cell. For communication to occur on the established ad-hoc network, the devices must have the same value assigned in the Cell field of the wireless interface settings. Furthermore, a seemingly related bug in the driver does not allow the Cell to be manually specified, disallowing a simple workaround to the cell splitting issue.

LG Nexus 4 The Nexus 4 does not work out-of-the-box because of a lack of IBSS support in the wireless chipset. These devices use a Murata SS2908001 802.11 a/b/g/n Wi-Fi chip. None of the suggested modifications to the kernel driver for the Murata SS2908001 802.11 a/b/g/n Wi-Fi chip successfully enabled IBSS support.

Samsung Galaxy SII The Samsung Galaxy SII was one of the only devices that worked out-of-the-box with the MANET Manager software. They are deployed with Android 2.3. After unlocking the device, the network communication worked very well. However due to requirements of other software necessary for our testing, we require the version of Android to be at least 4.0.3. After updating the Galaxy SII, the ad-hoc communication no longer worked.

Samsung Galaxy Tab 2 10.1 The Galaxy Tab 2 originally shipped with Android 4.0.3. Samsung has provided the device with updates to Android 4.1.1 and Android 4.2.2. None of these three firmware versions would support IBSS without modifications to the kernel. The chipset used in the Galaxy Tab 2 is the Broadcom 4330, which is the same chip as the Nexus 7. Therefore, the device has the same network driver as the Nexus 7, since the drivers are provided by the device manufacturers. Because of this, we could enable IBSS with the same modified kernel. Unfortunately, using the same chipset and driver also means that these devices suffer from the same cell splitting issue as the Nexus 7.

ASUS Eee Pad Transformer Prime The Transformer Prime ships with Android 4.0.3. IBSS support is successfully enabled (without any cell splitting issues) by performing a modification to the wireless chipset driver similar to the ASUS Nexus 7 modification. However, an even easier

solution is to update the device to Android 4.1.1 where IBSS is supported without any modification.

2.3.2 CyanogenMod

Alternatively, to manually patch the kernel wireless driver file, we identified that CyanogenMod, a customized version of the Android operating system, includes the necessary modifications to enable IBSS support for some Android devices, including the Nexus 7. CyanogenMod is designed to increase the performance and reliability of Android (see reference (*8*) for availability). Unfortunately, using CynanogenMod resulted in the same issues (cell splitting, unsuccessful manual Cell specification, etc.) as patching the kernels manually.

2.4 Optimized Link State Routing (OLSR) Protocol and MANET Manager

Once ad-hoc mode was enabled on our devices, the network exists but devices participating in an IBSS do not have knowledge of their neighbors. To enable devices to be aware of the network topology a routing protocol needs to be in place. The OLSR Protocol is a well-known routing protocol for mobile ad-hoc networks, but is not included in the Android operating system. Fortunately, there is an open source project called MANET Manager (*9*), which ports the OLSR daemon to Android devices. With this additional software we were able to successfully create a usable mobile ad-hoc network.

3. Software

In this section, we describe a software tool, MANET Tools, that we have developed to allow us to perform experiments to test latency, bandwidth, dynamic route configuration, and other aspects of our mobile ad-hoc network. This tool is implemented as a standard Android Application (app) that can be started like any other app that users are already familiar with by simply tapping the icon in the Application manager.

3.1 Android MANET Manager

Our MANET Tools app relies on an open source project, Android MANET Manager (*9*), to provide the low-level configuration necessary to establish and connect to an ad-hoc network. When MANET Tools is started, we connect to MANET Manager's ManetService to establish or connect to an existing ad-hoc network automatically. The user is not allowed to interact with the application (blocked by a ProgressDialog) until the network is successfully configured, as shown in figure 3.

Figure 3. A screenshot of our MANET Tools app automatically configuring the ad-hoc network connection.

Figure 4. A screenshot of the main MANET Tools interface.

3.2 MANET Tools Main Interface

A screenshot of our main interface is shown in figure 4. Capabilities include timing the transfer of one or multiple files to measure network throughput, sending a file repeatedly to simulate interference, measuring round trip time (RTT) in real time or logging it to compute time-averages, displaying a list of available devices, logging network traffic with 'tcpdump', and computing routes to available devices using 'traceroute'.

When the user rotates the device (from portrait mode to landscape mode, for example), the default Android behavior is to restart the Activity that clears all of the user's selections, settings, and member variables (effectively restarting the application). This is undesirable because the initial connection using MANET Manager takes several seconds, and it should be unnecessary to reconnect to the network just because the device orientation changed. To fix this, we use a Fragment, ActivitySelectorFragment, that is loaded by the MainActivity that maintains the object that handles the connection to the network via MANET Manager.

3.3 Reusable Utilities

There are some tasks that must be repeated in several different places throughout our app, so we have developed these tasks in a modular and reusable fashion.

3.3.1 Extracting Neighboring Devices from the Routing Table

In almost every task, the user must select a device present on the ad-hoc network (to send a file to, measure round trip time, etc.). To enable this selection with minimal code duplication, we provide a class, 'SelectReachableAddressDialogFragment', that produces a dialog from which the user can select a device that is available on the network. We use an AsyncTaskLoader to dynamically refresh the list of reachable devices. This list is obtained and updates to it are triggered from the ManetObserver's 'onPeersUpdated' callback.

3.3.2 Setting Up A Log File

When performing experiments on a network, it is often required to log the results. The user may wish to append to an existing log file, specify a new log file, or automatically create a log file based on the current time. We provide a Fragment, 'SelectLogFileFragment', to enable these features and conveniently reuse this code.

3.4 File Transfer

A simple measure of network performance is the time it takes to transfer a file of a given size from one device to another. As with any file transfer, the two main components are a transmitter and a receiver.

3.4.1 Transmitter

We provide an interface (shown in figure 5) for the user to select a file to transfer, select a destination device to send it to, select a log file, and select a number of times to transfer the file.

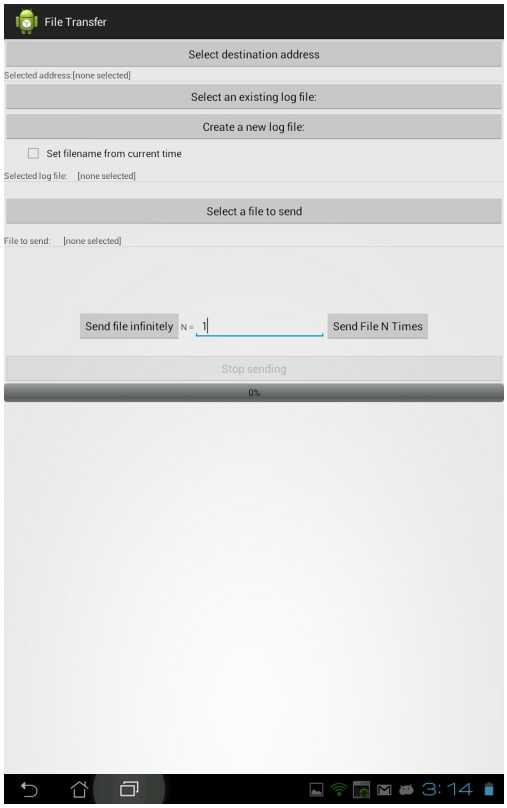

Figure 5. A screenshot of the user interface used to send files to another device.

The “Send file N Times” functionality is designed to allow recording of multiple instances of the same transfer in hopes of getting more accurate results via statistical averaging. The “Send file infinitely” functionality is designed to make a device act as an interfering device while measuring throughput from a different device to the same destination. These transfers are done in a different thread using an AsyncTask that we call ’TransferFileAsyncTask’.

In all cases, the file is sent in small chunks to allow us to track the progress of the transmission, as well as stop the transmission mid-file.

All logging is done on the transmitter side. When a file is sent successfully, we write the duration of the transfer to the specified log file.

As with the MainActivity, we remedy the Activity restart on orientation change by implementing the FileTransferActivity using the Fragment pattern so that transfer state persists across orientation change events. We save and restore User Interface (UI) elements using Bundles.

Creating Files To Send With A Specified Size It is often convenient to have a file of a particular size to perform throughput timing tests. For example, the larger the file, the more intra-file packet transfer times are averaged. The Linux command `dd` allows us to create a file of a particular size. The command `dd if=/dev/zero of=N_MB.dat bs=1M count=N` will create an N MB file. For example,
`dd if=/dev/zero of=10_MB.dat bs=1M count=10` will create a 10 MB file.

3.4.2 Receiver

To receive a file, a device must be listening for incoming connections. If we listen for connections and then when one is detected immediately start receiving a file, this disallows multiple simultaneous connections as the receiver cannot receive new connections while the file transfer is occuring. To remedy this, we have written a three-part file receiver. First, FileReceiverService is started immediately when MANET Tools is launched. It runs as a background service, allowing incoming file transfers even when the user has left the MANET Tools app. It starts a ConnectionHandler in a new thread. The ConnectionHandler listens for incoming connections, and when one is detected, it starts a FileReceiverThread to actually process the incoming file. The communication is done over a standard Transmission Control Protocol (TCP) socket connection. We first send the file name and the file size, and follow that with the file contents.

3.5 TCPDump

The 'tcpdump' command is a packet analyzer that is able to capture and describe packets arriving at a specified network interface. This binary comes with Android MANET Manager and can be found in '/data/data/org.span/bin'.

We make system calls to invoke TCPDump using Android's 'Runtime.exec()', and then log the output for later analysis. Being able to easily log these commands is useful in multiple scenarios. For example, the logs can be used to verify if algorithms are taking packets through the nodes on the network as expected.

In figure 6, we show an interface to setup and run a tcpdump command.

3.6 Traceroute

The 'traceroute' command prints the route that packets take between network nodes. When setting up experiments, being able to see the routes that packets are taking is extremely useful. Consider an experiment in which we want to send a file from device A to device C, but we want it

to first hop through a middle device B. To perform this experiment we must separate devices A and C by a great enough distance so that they are not in range of each other. However we want both of their signals to be in range of device B. To determine if this is the case, we make use of the built-in traceroute command. In the top half of figure 6, we show the interface to traceroute, which allows us to see the current path to a particular device on the ad-hoc network.

We simply parse the output (stdout) of the traceroute command to determine the route. An example output is shown below. Here, we have traced the route to 192.168.1.106. In this case, we can reach 192.168.1.106 through a 2-hop connection via 192.168.1.103.

```
traceroute to 192.168.1.106 (192.168.1.106), 30 hops max, 38 byte packets
 1  192.168.1.103 (192.168.1.103)  4.421 ms  2.751 ms  8.546 ms
 2  192.168.1.106 (192.168.1.106)  4.614 ms  2.871 ms  8.766 ms
```

The lines are structured as follows:

```
[traceroute information]
 [hop #]  [ip] ...
 [hop #]  [ip] ...
 ...
```

Therefore, we are interested in extracting the IPs immediately after the hop numbers starting on the second line. We do this by splitting the lines on spaces, and taking the 4th (id = 3) field, as there is a leading space, and two spaces between the hop number and the ip. In the example above, our graphical user interface (GUI) displays

```
192.168.1.103 -> 192.168.1.106
```

indicating that we reached 192.168.1.106 via 192.168.1.103. This chain can be as long as necessary to describe the route (e.g., A -> B -> C -> D -> ... -> destination).

3.7 Time (RTT) Computation

We compute the RTT of packets between nodes using the 'ping' command. To run a single ping and extract the RTT, we would typically run the command: 'ping -W 1-c 1 -q mAddress | grep rtt | cut -d "/" -f5'. In this command, '-W 1' indicates to wait for no more than 1 second before

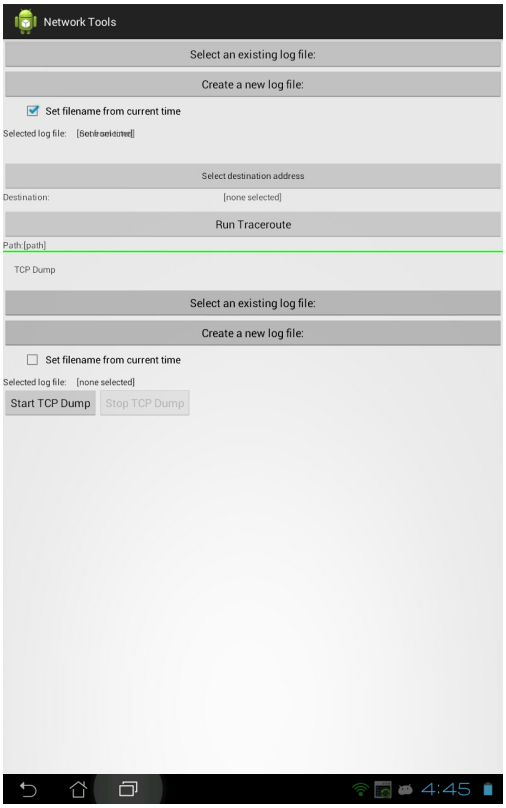

Figure 6. The top half of the NetworkTools activity allows the user to setup, run, and view the result of a traceroute. The bottom half of the NetworkTools activity allows the user to log TCP traffic to a file using tcpdump.

timing out, and '-c 1' indicates to send only one packet. The rest of the command is parsing the output of the line containing the average RTT. To see what this is doing, we look at a sample output of 'ping -c 1 -q address':

```
/home/doriad $ ping -W 1 -c 10 -q [address]
PING [address] 56(84) bytes of data.

--- [address] ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9010ms
rtt min/avg/max/mdev = 0.962/1.104/1.353/0.115 ms
```

We see that the value we are after (the average RTT), 1.104, is in the 5th block delimited by the '/' character. The Linux command 'cut' allows us to parse strings and extract blocks. By grepping the output of ping for 'rtt', we obtain the line with the RTT values. By piping this line to 'cut' with instructions to delimit on '/' and extract the 5th block, we obtain the value of interest.

Unfortunately, this does not work directly in a Runtime.exec() call because pipes are not handled correctly inside exec(). Rather than pass the command directly, we have to create a string that instructs exec() to pass the command to 'sh' as a parameter, as follows:

```
String[] fullCommand = { "sh", "-c", commandToRun };
```

Additionally, we have to escape quotes in the construction of the command with a backslash. This makes the final 'commandToRun':

```
String commandToRun = pingBinary + " -W " + timeout + " -c 1 -q " +
mAddress + " | grep rtt | cut -d \"/\" -f5";
```

In figure 7, we show our user interface to compute and log round trip time to a specified device.

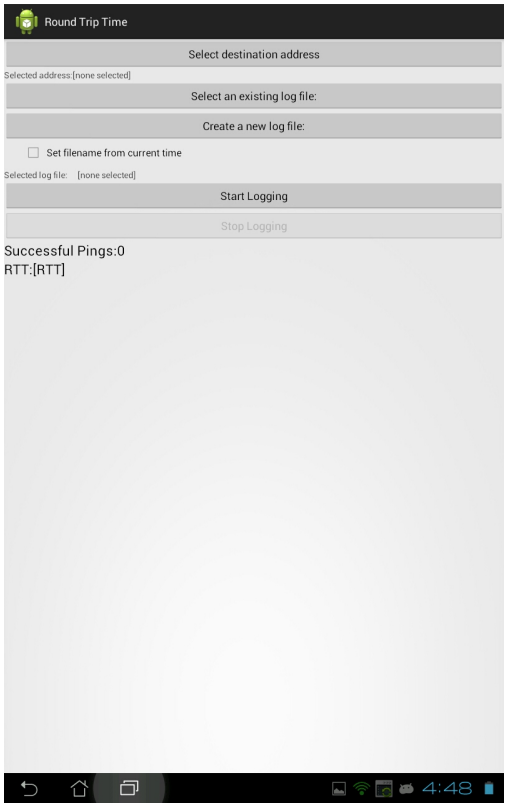

Figure 7. The RTT activity allows the user to compute and log the round trip time to a device on the ad-hoc network.

4. Summary and Conclusions

In this paper we have described the steps necessary to construct a mobile ad-hoc network on Android devices. We discussed several software tools to enable us to more easily setup and conduct experiments studying routing, network delay, and network throughput. We showed how these experiments are critical to future research in areas related to mobile ad-hoc networks.

5. References

1. Kwasinski, A.; Weaver, W.; Chapman, P.; Krein, P. Telecommunications Power Plant Damage Assessment for Hurricane Katrina x2013; Site Survey and Follow-Up Results. *Systems Journal, IEEE* **2009**, *3* (3), 277-287.

2. Johnson, D.; Hancke, G. Comparison of two routing metrics in {OLSR} on a grid based mesh network. *Ad Hoc Networks* **2009**, *7* (2), 374 - 387.

3. Swenson, B.; Doria, D.; Sookoor, T. *Simulation and Evaluation of Computation Offloading Algorithms in Battlefield Scenarios*; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD To be published in 2014.

4. Kerr, D. Android beats iOS 5-to-1 in Q3 smartphone market share. http://news.cnet.com/8301-1035_3-57544131-94/android-beats-ios-5-to-1-in-q3-smartphone-market-share/, Accessed January 2014.

5. Magne, J. "Comparison of the IEEE 802.11, 802.15.1, 802.15.4 and 802.15.6 wireless standards", Master's thesis, University of Stavanger, 2007.

6. Wi-Fi Alliance, Wi-Fi Alliance Member Symposium. www.wi-fi.org/, accessed January 2014.

7. Lynn, S. Wi-Fi Direct: What You Need To Know. http://www.pcmag.com/article2/0,2817,2371413,00.asp, 2010.

8. CyanogenMod Android Operating System Replacement. www.cyanogenmod.org, 2014.

9. MANET Manager Android Software. https://github.com/ProjectSPAN/android-manet-manager, 2013 Accessed: 2013-09-30.

List of Symbols, Abbreviations, and Acronyms

app	Application
BSS	Basic Service Set
DARPA	Defense Advanced Research Projects Agency
GUI	graphical user interface
HPC	High Performance Computer
IBSS	Independent Basic Service Set
MAC	Media Access Control
OLSR	Optimized Link State Routing
P2P	Peer-to-Peer
PRNET	Packet Radio Network
RTT	round trip time
TCP	Transmission Control Protocol
UI	User Interface

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
6 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIH S DAVID BRUNO DAVID DORIA TAMIM SOOKOOR DALE SHIRES RONDA TAYLOR SONG PARK